LA-UR-99-6062

XTV COMPLETION REPORT

by

Russell C. Johns

Los Alamos National Laboratory

March 2000

Rev. 4.0

XTV COMPLETION REPORT

by

Russell C. Johns

ABSTRACT

This document describes the internal format used by the Transient Reactor Analysis Code (TRAC) and a related set of postprocessing tools for graphics output of the model information generated by TRAC. This is the fourth major revision of the format. Previous versions have used a combination of text and platform-dependent-binary output to convey the information. This release now uses External Data Representation encoding throughout the file to generate portable, easily parsed data for use in any of the available postprocessors. This version represents the consolidation of two major branches in the file format to support the needs of different sponsors. This consolidated format meets the needs of each sponsor and provides additional features (such as templates) to simplify the creation of intuitive graphical user interfaces.

1.0. INTRODUCTION

X-TRAC-View, an X-Windows-based graphical postprocessor for the Transient Reactor Analysis Code (TRAC), previously used a separate ASCII file (xtvgr.t, or file.xtvt) to describe the contents and format of the binary graphics (xtvgr.b, or file.xtvb) output. When the X-TRAC-View (XTV) file content was expanded to include full TRCGRF equivalency, the format of the XTV file changed. Foremost among those changes was the consolidation of the two files into one container file (trcxtv or file.xtv), which subsequently has been modified to encode all parts of the file in eXternal Data Representation (XDR). XDR is an Applications Programming Interface (API) that forms part of Sun Microsystems' Open Network Computing group Remote Procedure Call API. This new coding improves platform independence and merges better with existing United States Nuclear Regulatory Commission graphics files. In the current version, 4.0, graphics templates support has been added to simplify the use of variables of different dimensions within the same component. This document describes the output format of the XTV graphics file (see the TRAC Programmer's Manual[1] and User's Manual[2] for the contents of the XTV file and the TRAC implementation, respectively.)

The major changes since Version 3.0 include the following:

- Addition of templates for display.
- Encoding of the entire header in XDR.
- Consolidation of the junction structure back to a single implementation for all general types.
- Addition of matrix types.
- Addition of a format revision field in the block header.
- Output of all floating-point header values in 8-byte format.

2.0. SUMMARY OF XTV HEADER FORMAT

The XTV file consists of two sections: an XDR-encoded header (catalog) and an XDR-encoded data section. The header section comprises many modules, which in turn comprise one or more blocks, which may comprise one or more sub-blocks. Each of the most fundamental blocks or sub-blocks has a subroutine associated with its input and output. This subroutine is written so that it can be used either for reading or writing the block data. The subroutine is application-independent in that it should serve well for use in TRAC, XTV, XMGR5 or another postprocessor. Higher-level subroutines that are specific to an application, such as TRAC, will select the mode of operation (reading or writing) and process the information as appropriate to the application.

The header comprises a starting module, which contains parameters and information that pertain to the file as a whole, and many component modules that specify how to represent a particular component and what variables are provided for that component.

2.1. Starting Module

The starting module has increased dramatically in content from previous versions. The original starting module was a one-line title taken from the first title card supplied through TRACIN. In Version 2.1, the starting module content was augmented to support version numbers, as well as the title on the first line. In Version 3.0, the starting block added flags for format and precision, as well as debugging information such as date and time of creation, machine name and platform type, and additional units. Finally, in Version 4.0, the starting module added dimensional information for the number of components, total number of variables, and number of timesteps output to the file.

The starting module consists of the start block and many optional units blocks. The start block contains all of the global file dimensions, the problem title, and the debugging information. The units blocks contain information needed to use TRAC's English units system in the postprocessor. Under the current TRAC architecture, these blocks typically will be generated only when English units are in use, although they can be input for SI units.

2.1.1. Start Block

Because of the nature of the XDR encoding, parsing of the information is considerably reduced in the start block. String parsing is required only in three fields in the revised XDR header and can be omitted entirely in less-than-ideal circumstances. Many other fields need checking; however, they are delineated by the XDR encoding and so are easier to use. The fields that should be parsed are the XTV identification string, the format string, and the units system string.

The first value in the file is the XTV identification string, which should be parsed to ensure that this is an XTV file. For verification purposes, only the first three letters need to be checked, along with the major and minor version numbers. The remainder of the string is information to identify uniquely the program and, more specifically, what version of that program created the file. Previously, there was only one XTV file generator, and thus, the XTV identifier was the string "XTV-TRAC". This, when combined with a version number, identified that this was an XTV file and that it could be read by XTV if the version number of the graphical user interface (GUI) was equal to or greater than the version number in the file. There are now multiple versions of TRAC that output XTV format graphics files. The XTV identification string is now "XTV-TRAC/f90" to distinguish it from previous versions.

Additionally, there was no way to determine exactly which version of TRAC was used to generate the file. That is, although the version number reflected the XTV level in TRAC, it gave

no indication of how many other changes in TRAC had been made since XTV last was updated. Because of this, we have augmented the version number with a third value or revision number. The new version number consists of the version of XTV needed to display the file, followed by a revision number to uniquely identify the version of TRAC used. The revision number is generated by incrementing the number by 1 for each version of TRAC since the latest change to the XTV file output. For example, the version of XTV that will be able to read this new format will be designated 4.0; thus, the version number presented in the header file written by TRAC will be Version 4.0.0. If we make two revisions to TRAC, the version identifier written by TRAC would become Version 4.0.2.

The format string should be parsed to ensure that this file is in the standard format, “MUX”. The XMGR5 postprocessor also understands a second format, “DEMUX”, where the graphics data are stored by variable rather than by time. TRAC cannot generate a graphics file in the DEMUX format.

The last string that needs to be parsed is the units system string. This value provides the user with the system of units used in the file (English or SI units). This is particularly useful if the postprocessor understands the TRAC English units label system, where each value has a units-type label.

Many variables also are necessary to read the remainder of the file correctly. The variable *xtvRes* contains a flag to indicate whether the data block uses single-precision (32-bit or 4-byte) or double-precision (64-bit or 8-byte) values. Typical files use single-precision values, but 8-byte values have been added for debugging purposes.

The variable *nUnits* indicates the number of units blocks; if the TRAC English units system is not supported in the postprocessor, these blocks may be skipped over, but they follow immediately after this block.

The variable *nComp* indicates the number of component modules that are included after the starting module. This variable provides (1) the information needed for dimensioning the space for the component data in the host application and (2) the input control to separate the header segment from the data segment.

Several additional dimensional variables may be useful to the postprocessor: *nSVar*, *nDVar*, *nSChannels*, *nDChannels*, and *nPoints*. *nSVar* and *nDVar* contain the total number of static and dynamic variables defined in all components. This is useful in postprocessors that implement global variable tables. *nSChannels* and *nDChannels* contain the total number of static and dynamic channels, respectively. A channel represents a variable at a particular cell or cell face, and thus, the number of channels is equivalent to the number of values in a particular edit. *nPoints* contains the number of timesteps or datapoints contained in the file. To facilitate runtime visualization, *nPoints* is updated at the end of each additional timestep until the completion of a run. Thus, nPoints always contains the number of complete edits on the file.

The remaining values either identify the run or the conditions under which the run was generated, which can be useful for both the user and the debugger.

2.1.2. Units Blocks

The units blocks enable intelligent units description and conversion. TRAC has an extensive understanding of the type of any particular variable (e.g., *vol* is type volume and *vv* is type velocity) and has facilities for augmenting this list for the particular problem. TRAC will supply the postprocessor with the user-defined units types as part of the XTV file so that the postprocessor has a full and complete list of all fundamental units types. The start block provides the number of additional units being supplied to the postprocessor and the units system utilized

in the file. The units blocks provide SI and English units labels for each units type, as well as the necessary conversion information. Postprocessors that do not want to use the English units system provided can use the labels supplied with each variable for plot output. Each units block provides the information for one unit type, so there will be as many blocks as additional units types specified in the start block.

2.2. Component Module

The component module consists of as many as nine different blocks: a generic component parameters block, three types of graphic display template blocks, dynamically sized axis blocks, junction blocks, leg blocks, an auxiliary component block, and variable definition blocks. The only block that must be present is the component parameter block, although it is anticipated that all components will have variable definition blocks.

The generic component parameter block is the same for all components. It contains the component label information, as well as the number of each type of additional block supplied. Diagnostic information also is included to verify some of the additional block information.

The graphic display template blocks come in three forms: one dimensional (1D), two dimensional (2D), and three dimensional (3D); scalar variables require no template. The graphics display templates contain both the physical dimensions of the variable and all of the information needed to represent that variable graphically in a GUI. Much of this information can be discarded if the postprocessor is concerned only with plot generation. Each component must have a template that matches the rank of the component (purely scalar components have no associated templates), but may have as many other templates as needed. The rank of the component is defined as the highest rank of any of its variables (e.g., a Pipe may be either a 1D or 2D component, depending on whether it has wall heat conduction activated).

The dynamically sized axis blocks convey how a particular axis is dynamically sized. This will be referenced by one or more of the graphics display templates. It also will provide the variable that contains the positions of the data output. There can be as many dynamic axis blocks as needed, but it is evident that only three at most may be referenced by any single variable, and those blocks will be used for each and every timestep.

Junction blocks specify the connectivity between components. A standard 3D block is used; lower-dimensional components should supply 0 for the index of the unused dimensions. Scalar components have no junctions, although there is no longer any reason they cannot have them.

The leg blocks supply information used to partition the component into a main segment and many secondary segments. The model employed in XTV is somewhat more general than what is currently in TRAC; legs can exist on any 1D, 2D, or 3D component, and there can be any number of them. The data structure is defined to break on the last axis [x/I for 1D, J (typically z) for 2D, and K(z) in 3D]. Each leg is defined by three values: the index (x/z) of the first cell of the side leg, the index of the last cell of the side leg, and the index of the cell to which this side leg attaches, which could be on the main leg or another side leg.

The auxiliary component block is used as a way to allow component-specific (Pipe, Tee, Plenum, etc.) information into the generic format. The block is nonessential information that a GUI or other postprocessor can use to enhance the display of a particular component but can be safely ignored to treat the component in a more generic way. It is the only block that is truly TRAC-component specific. The 14 TRAC components map to only 4 XTV component types.

The primary advantage of this system is the reduction in effort required in the postprocessor to add additional component types, such as the channel component. This component should be able

to be classified as a variety of 1D component, which can be directly read and displayed by the current postprocessors. If desired, special display routines can be added later, but a minimum level of functionality is available as soon as it is created.

Lastly, the variable definition block provides the information specific to each variable supplied for that component and any display information that is variable specific. This is supplied through many variable attributes, as well as through a reference to the appropriate graphics display template.

2.2.1. Component Parameters Block

The component parameters block contains nearly all of the information common to all components. Among the most important of these parameters is the component-type identifier or class. The component-type identifier determines the default functionality of the component in the postprocessor. The variation in component behavior comes through feature differences. For example, the behavior of Pipes, Tees, and Plenums can be seen in the number of junctions and legs of each. A Pipe has two junctions, a Tee has three junctions and one leg, and a Plenum has three or more junctions and no legs. This allows the same generic display coding to support all three 1D components and yet look and behave differently.

The next most important items are the component identification number (ID number) and substructure ID number. Unlike previous versions that could not define a component precisely with a single numbering system, the substructure ID number enables all XTV components to be identified uniquely.

2.2.2. Graphics Display Template Block

The graphics display template block contains the information needed to create the graphics template or visual tile for a particular class of variables in XTV. Unlike previous versions of XTV, there is a template for each class of variable. A template can handle both cell-center and cell-edge variations in variables, but different templates are required for variables that differ in either array rank (1D, 2D, or 3D) or array dimension. Each template includes the total number of cells or nodes, the number of cells along each axis, the cell length and width, and a reference to the appropriate dynamic axis structure (if used). The component now stores the junction and leg information. This structure varies significantly from no detail (0D) to high detail (3D). See the detailed description below for specific information.

2.2.3. Auxiliary Component Structure Block

The auxiliary component block is used to allow component-specific information into the generic format. This nonessential information can be used by a GUI or other postprocessor to enhance the display of a particular component but safely can be ignored to treat the component more generically. Currently, the only auxiliary component structure is for the Plenum component and contains the computational lengths of each junction. This had no way of being carried in the generic format because the Plenum is a single-celled component. Later, this information could be used to draw a more accurate model of the Plenum cell than the current square.

2.2.4. Variable Definition Block

The variable definition block now contains only the actual variable definitions. Previous versions had a variable-count sub-block, but that information was consolidated into the component parameters block above. Each individual variable definition block contains the name, description, units type and label, length, template index, and series of five variable characteristics or attributes.

Before Version 3.0, each variable had a midsized name and size identifier. This has been augmented with a short name, as well as a potentially longer description (the former name), which allows for easier scripting and selection from a more comprehensive list.

The units label can be used by programs that are not units-type savvy, whereas the units-type identifier can be used by those wanting to perform units conversion.

Both for internal checking and easy access, the length gives the exact number of elements output. This number also can be calculated from the variable attributes and template information.

As described above, the template explains how to create a visual display for the variable and provides the physical extent of the variable. The template index references the specific template that applies to the variable.

The variable attributes are responsible for defining all of the important variable characteristics. The five variable attributes are the dimension/position (hybrid), frequency, color map, vector, and special options.

The dimension/position attribute is used to allow several different dimensions (0D, 1D, 2D, and 3D, as applicable to the component type) within a component. This attribute is hybridized with position (cell center or cell face) for the combinations used in TRAC.

Frequency currently is time-dependent or time-independent but could be expanded to allow for varying rates of data output, depending on the class of variables. Some variables could be output every dump, whereas others could be output every other dump. Time-independent variables (e.g., *vol*) are output immediately after their specification and are not found in the data section of the file.

Color mapping determines whether water colors (blue) or hot colors (red) are used to map the display tiles. This mapping provides easy visualization of the quantity when both flow conditions and heat/energy quantities are being output from the same component.

The vector attribute is used to support higher levels of associations than a single value per cell. The only vector option currently used is the 3D vector association, where three variables representing a particular behavior along a selected axis are associated to generate a better concept of how the model is behaving. The attribute is designed to handle other complex associations, such as matrices, intrinsic vectors, and tensor associations.

“Special options” is the catchall for the remaining options. Currently, there are only two special options: inset display and unlisted. Both help to inform the GUI of potential uses of the variable and are completely optional (see the table at the end of the document for a more thorough description of each item).

2.3. End-of-Header Block

There is no longer an end-of-header block. The end can be determined by the completion of the component modules or from the dataStart value in the start block described in Section 2.1.1.

3.0. SUMMARY OF XTV DATA FORMAT

3.1. Header/Data Interface

The binary data follow immediately after the header file. Because the entire file is encoded in XDR, there is no need to reopen the XTV file after reading the header. The data are in the same order as the header presents them.

3.2. Data Format

Each timestep edit comprises a short binary header and subsequent component data blocks. Each header comprises the string “DATA”, followed by a revision stamp, and the size of the block. After the generic block delimiter, the data blocks contain two additional values, a second size of block parameter, and the current problem time. The second size of block parameter allows the entire datablock to be read in to a single array, if desired, through the use of an XDR array read.

Each component data block will be in the order presented in the index. Multidimensional arrays are output as they occur naturally in FORTRAN (i.e., the first index changes the most rapidly (Row-Major), followed by the second index, then the third, etc.).

To read in all values of a particular variable, typically each successive timestep edit would be read in, the value desired would be obtained, and then the remaining data would be discarded. Then the next edit would be read in and the procedure repeated. This is most particularly true if the data are compressed or have dynamically dimensioned output, in which case there is little choice. If the data are not compressed and there are no dynamically dimensioned variables before the desired one, then the offset can be computed into the edit and stepped through with little processing. Currently, the heat structures output fixed-length, dynamically used (some space is undefined/unused) variables; thus, each timestep edit is exactly the same length. It may be advantageous in terms of space to switch to dynamically dimensioned variables, in which case the access time will be increased significantly.

To read in multiple values at the same timestep, e.g., for the void fraction profile in a Pipe, the data block must be stepped through to the relevant timestep and the values extracted from the array alpha(x). Typically, batch files and users access one variable at a time; thus, all elements desired generally will be in the same array.

4.0. DETAILED HEADER FORMAT

4.1. Component-Independent Information

4.1.1. Starting Block

Item No.	Name	Type	Length	Description
1	hdrString	xdr_bytes	lenHdrStr	XTV identification string
2	xtvMajorV	xdr_long	1	Version major number (e.g., 4)
3	xtvMinorV	xdr_long	1	Version minor number (e.g., 0)
4	revNumber	xdr_long	1	XTV revision number
5	xtvRes	xdr_long	1	Size of float output (four or eight)
6	nUnits	xdr_long	1	Number of units information blocks output
7	nComp	xdr_long	1	Number of components output (XTV comps)
8	nSVar	xdr_long	1	Total number of static variables output
9	nDVar	xdr_long	1	Total number of dynamic variables output
10	nSChannels	xdr_long	1	Total number of static data channels output (number of data values output as variables in header)

11	nDChannels	xdr_long	1	Total number of dynamic data channels output (number of values output/timestep in data section)

Item No.	Name	Type	Length	Description
12	dataStart	xdr_long	1	Location of first data edit
13	dataLen	xdr_long	1	Length of each data edit
14	nPoints	xdr_long	1	Number of datapoints in file
15	spare1	xdr_long	1	Reserved expansion room
16	spare2	xdr_long	1	Reserved expansion room
17	spare3	xdr_long	1	Reserved expansion room
18	spare4	xdr_long	1	Reserved expansion room
19	fmtString	xdr_string	lenFmtStr	String containing possible format code
20	unitsSys	xdr_string	lenUnitsSys	Units system identifier
21	sysName	xdr_string	80	Name of the machine (e.g., starTrac)
22	osString	xdr_string	80	Operating system name (e.g., IRIX)
23	sDate	xdr_string	80	Starting date of run
24	sTime	xdr_string	80	Start time for run
25	title	xdr_string	lenTitle	User-supplied run title

Item 1: File identification string
"XTV-TRAC/f90" is used to identify that this is an XTV file written by the Fortran 90 version of TRAC-M.

Items 2–4: Version number of format *n.m.r*
n Major XTV version.
m Minor XTV version.
r Revision number = current TRAC version number—TRAC version number when XTV was last changed for input handling.

Item 5: Resolution flag
XtvRes determines whether the data section contains single- or double-precision values. "4" indicates single-precision floats (xdr_float), whereas "8" indicates double-precision floats (xdr_double).

Item 6: Number of additional units entries
The number of entries indicates the number of units information blocks defined below that are supplied. The system in use is supplied in Item 16, unitsSys.

Item 7: Number of XTV components
An XTV component differs from a TRAC component in two areas. First, XTV defines four generic component types: scalar, 1D, 2D, and 3D. Pipes, Tees, Pumps, Vessels, etc., all are recast as one of these fundamental XTV types. Second, in addition to outputting general problem information and control system data as components, individual rods and slabs are defined in XTV as their own components rather than as parts of the same component.

Items 8–9: Total variable counts
These two variables represent the total number of variables output in the header and in the data sections, respectively. The sum of nSVar and nDVar is the sum of all variables output anywhere in the graphics file.

Items 10–11: Number of data channels
These two variables represent the total number of data values output in the header and in a single data edit, respectively. These differ from items 8 and 9 in that these include not only the number of variables, but also the number of

cells for each variable. These values are used by xmgr5, which outputs each cell location as an independent location rather than outputting each component. Note that nDChannels also counts the problem time as a data channel so that if all variables were in unicellular components, nDChannels would be equal to nDVar plus one.

Items 12–13: Start and length of data blocks
dataStart is a file pointer that can be used to jump to the immediate start of the data edits. This can be used in conjunction with dataLen to jump to any specific edit requested without reading any more of the header than the start block. This is used for TRAC' append capability.

Item 14: Number of datapoints
This variable contains the total number of graphics dumps in the file. It is updated at the end of each edit and so guarantees the validity of as many edits. This is used by the SNAP runtime editor to read the appropriate level of information during execution.

Items 15–18: Spare data locations
These items are placeholders for items that may need to be added later.

Item 19: Format code for file

MUX TRAC can only output format “MUX”; data are present in multiplexed format (i.e., all the variables are output for a given time, then the next set of variables is output for a different time).

DEMUX XMGR5 understands a second format, “DEMUX”, or demultiplexed data. All times for a single variable are output, followed by all times for a second variable, etc.

Item 20: Units system and number of entries
The units system can be either “SI” or “ENG”. This indicates that the data section uses SI or English units for the values. The number of entries indicates the number of units information blocks defined below that are supplied.

Items 21–22: Machine and OS identifier
These items aid in the debugging process. They can be obtained through the gethostname() and uname system calls.

Items 23–25: Date, time, and title
These items provide user convenience in identifying the details of the run. The title is taken from the first title card provided to TRAC in the input specification.

4.1.2. Units Block

Item No.	Name	Type	Length	Description
1	labun	xdr_string	lenLabun	Units type label (e.g., lutemp)
2	siLab	xdr_string	lenSiLab	SI label
3	engLab	xdr_string	lenEngLab	English units label
4	factor	xdr_double	1	Conversion factor
5	offset	xdr_double	1	Conversion offset

Item 1: Labun

This is the standard TRAC Units type label defined in the English units module.

Items 2–3: SI and English units labels
These are the SI and English units labels, respectively, to be used for axis labels or personal reference.

Items 4–5: Conversion factors
These two values can be used to convert from one system to the other by means of the equation Eng = SI * factor + offset.

4.2. Component Module

COMPONENT PARAMETER BLOCK
DYNAMICALLY SIZED AXIS BLOCKS (OPTIONAL)
GRAPHICS DISPLAY TEMPLATE BLOCKS (FOR ARRAY VARIABLES)
JUNCTION INFORMATION BLOCKS (OPTIONAL)
LEG INFORMATION BLOCKS (OPTIONAL)
AUXILIARY COMPONENT STRUCTURE BLOCK (OPTIONAL)
VARIABLE DEFINITION BLOCKS

4.2.1. Component Parameter Block

Item No.	Name	Type	Length	Description
1	compId	xdr_long	1	Component ID number from TRAC
2	compSsId	xdr_long	1	Substructure ID number
3	cType	xdr_string	8	TRAC Component type (e.g., Pipe)
4	cTitle	xdr_string	32	User-supplied component title
5	cDim	xdr_long	1	Dimension of the component

Item No.	Name	Type	Length	Description
6	nTempl	xdr_long	1	Number of graphics display templates
7	nJun	xdr_long	1	Number of junctions
8	nLegs	xdr_long	1	Number of legs
9	nSVar	xdr_long	1	Number of static variables
10	nDVar	xdr_long	1	Number of dynamic variables
11	nVect	xdr_long	1	Number of vector associations
12	nChild	xdr_long	1	Number of children
13	nDynAx	xdr_long	1	Number of dynamically sized axes
14	auxStrT	xdr_string	lenAuxStrT	Type of Aux structure to follow

Item 1: Component ID number
This is the unique component ID number (input variable *num*) given to the component on input.

Item 2: Component substructure ID number
This number is 0 for all non-HTSTR input components. In the heat structures, compSsId 0 gives general component-wide information and compSsId1+ gives information particular to the rod or slab with that number. Additionally, component 0-0 (compId = 0, compSsId = 0) is general problem information (dtmax, dprmax, etc.), component 0-1 contains the signal variables, 0-2 contains the control blocks, and 0-3 contains the trips.

Item 3: Component name
This is the standard TRAC component name (e.g., Pipe, Tee, Rod, etc.), with an eight-character maximum.

Item 4: Component title
This is the user-supplied label provided in input.

Item 5: Component dimension or class
This variable is used for error checking and general processing. All of the TRAC components have been divided into four generic groups or classes: 0D (scalar), 1D, 2D, or 3D. This determination is made by the highest-ranking variable present in the component.

Item 6: Number of graphics display template blocks
This variable is used to both allocate storage for the graphics display templates and control the file reading of these templates that follow. As discussed above, each variable is associated with a template that provides the postprocessor with the dimensions and other information to produce a graphical representation of that variable. Templates can handle both cell-center and cell-face values, but different templates are required for variables of differing dimension or rank. The template blocks follow any dynamic axis blocks that may be present.

Item 7: Number of junctions
This variable both allocates storage for and controls the file reading for these junction information blocks that follow. The junction information block is a generic structure that has indices for up to three separate axes; thus, the same

block is used for 1D, 2D, and 3D components. The junction blocks follow any template blocks that are present.

Item 8: Number of legs
This variable both allocates storage for and controls the file reading of the leg information blocks that follow. As mentioned previously, XTV's leg capabilities are greater than those currently used in TRAC—they can be 1D, 2D, or 3D and can have more than one leg per component. The leg block follows the junction block.

Item 9: Number of static variables
This variable both allocates storage for and controls the file reading of the Variable definition blocks that follow. Static variables carry the associated data immediately following the variable definition (before the next variable definition) because the data are time-independent for that calculation.

Item 10: Number of dynamic variables
This variable both allocates storage for and controls the file reading of the Variable definition blocks that follow. Dynamic are present at each graphics edit.

Item 11: Number of vector associations
This variable allocates storage for the additional vector association data that are carried in the variable definition blocks. Some variables (e.g., radial liquid velocity) form part of a vector association that provides additional information when combined into a graphical vector. This information is provided in advance of the variable definition blocks to assist the GUI in displaying these vectors.

Item 12: Number of child components
This variable checks for errors and is the number of substructure components defined for this component. This means it is currently non-zero only for the generic problem information component and the HTSTR general information components.

Item 13: Number of dynamically sized axes
This gives the number of dynamic axes defined for the component. Each dynamic axis defined for the component must have a dynamic axis block provided, as defined below. The dynamic axis blocks immediately follow this block.

Item 14: Auxiliary component structures
This defines the type of auxiliary component structure defined below. For almost all components, this will be "AUX_NONE". The auxiliary component structure block follows the junction and leg blocks.

4.2.2. Dynamically Sized Axis Blocks

Dynamically sized axes are used on components such as TRAC heat structures, where one or more of the axes are typically dynamically sized in a specified manner with a maximum number of elements. This assists in generating proper templates for the associated variables.

Item No.	Name	Type	Length	Description
1	dsAx	xdr_bytes	1	Code for axis
2	varType	xdr_string	2	Dynamic sizing attribute
3	sVarName	xdr_string	sVarNameLen	Scaling variable name
4	lVarName	xdr_string	lVarNameLen	Dimensioning variable name
5	vMax	xdr_long	1	Maximum number of cells

Item 1: dsAx
Code ("I", "J", "K") representing the axis being dynamically sized.

Item 2: varType
Variable dynamic sizing attribute as defined in variable attributes section for the variable that determines the size of the other variables.

Item 3: sVarName
Name of the variable that controls the scale of the dynamically sized variables. Note that this ideally should come before the dynamically dimensioned variables and must come before any variables where the output length varies. (Current variables are output at a fixed length, but only a fraction of them contain real data.)

Item 4: lVarName
Name of the variable that controls the length of the dynamically sized variables. Note that this must come before the dynamically dimensioned variables and before any variables where the output length varies. (Current variables are output at a fixed length, but only a fraction of them contain real data.)

Item 5: Maximum number of cells
Maximum number of elements present. The axis dimensional information should be given for this number of cells.

4.2.3. Graphics Display Template Blocks

Graphics display template blocks exist for each class of variable in the component. A variable class is determined by both the number of dimensions for a variable and each specific dimension.

4.2.3.1. 1D Graphics Display Template

The 1D graphics block consists of the following sub-blocks:

- Dimensions of the 1D graphics template
- 1D parameter arrays sub-block

4.2.3.1.1. Dimensions of the 1D Graphics Template

Item No.	Name	Type	Length	Description
1	nCells	xdr_long	1	Number of cells in this component
2	dynAxI	xdr_long	1	Index of the relevant dynamic axis structure (0 = NONE)

Item 1: nCells
This variable contains the number of values along the flow direction.

Item 2: dynAxI
This variable contains the index of the relevant dynamic axis structure for the I axis. Input 0 for NONE, 1 for the first structure, etc.

4.2.3.1.2. 1D Parameter Arrays Sub-Block

Item No.	Name	Type	Length	Description
1	fI	xdr_double	nCells +1	Coordinates of the cell faces
2	grav	xdr_double	nCells +1	Gravity vectors for each cell face
3	fa	xdr_double	nCells +1	Flow area for each cell face

Item 1: fI
This array contains the coordinates of each face along the I axis.

Item 2: grav
This array contains the cosine of the gravity vector at each face.

Item 3: fa
This array contains the cross-sectional flow area for each face along the I axis.

4.2.3.2. 2D Graphics Display Template

The 2D graphics template comprises the following sub-blocks:

- Dimensions of the 2D graphics template
- 2D parameter arrays

4.2.3.2.1. Dimensions of the 2D Graphics Template

Item No.	Name	Type	Length	Description
1	nCells	xdr_long	1	Number of cells in this component
2	nCellI	xdr_long	1	Dimension of the I axis
3	nCellJ	xdr_long	1	Dimension of the J axis
4	dynAxI	xdr_long	1	Dynamic axis index for the I axis
5	dynAxJ	xdr_long	1	Dynamic axis index for the J axis
6	coordSys	xdr_string	8	Coordinate system code

Item 1: nCells
This variable contains the number of cells in the component.

Item 2: nCellI
This variable contains the number of cells along the I (x, or radial) axis.

Item 3: nCellJ
This variable contains the number of cells along the J (y, or axial) axis.

Item 4: dynAxI
This variable contains the index of the relevant dynamic axis structure for the I axis. Input 0 for NONE, 1 for the first structure, etc.

Item 5: dynAxJ
This variable contains the index of the relevant dynamic axis structure for the J axis. Input 0 for NONE, 1 for the first structure, etc.

Item 6: coordSys
This variable contains the code for coordinate system. Valid codes are

CART2D	Cartesian (x,y)
CYLrt	Cylindrical (r,θ)
CYLrz	Cylindrical (r,z)
CYLtz	Cylindrical (θ,z)

4.2.3.2.2. 2D Parameter Arrays

Item No.	Name	Type	Length	Description
1	fI	xdr_double	nCellI +1	Coordinates of the cell faces on I
2	fJ	xdr_double	nCellJ +1	Coordinates of the cell faces on J
3	grav	xdr_double	nCellJ +1	Gravity cosines for cell faces on J

Item 1: fI
This array contains the coordinates of each face along the I axis.

Item 2: fJ
This array contains the coordinates of each face along the J axis.

Item 3: grav
This array contains the cosine of gravity vector for each face along the J axis. The I axis is assumed to be orthogonal.

4.2.3.3. 3D Graphics Display Template

The 3D graphics template consists of the following sub-blocks:

- Dimensions of the 3D graphics template
- 3D parameter arrays

4.2.3.3.1. Dimensions of the 3D Graphics Template

Item No.	Name	Type	Length	Description
1	nCells	xdr_long	1	Number of cells in this component
2	nCellI	xdr_long	1	Dimension of the I axis
3	nCellJ	xdr_long	1	Dimension of the J axis
4	nCellK	xdr_long	1	Dimension of the K axis
5	dynAxI	xdr_long	1	Dynamic axis index for the I axis
6	dynAxJ	xdr_long	1	Dynamic axis index for the J axis
7	dynAxK	xdr_long	1	Dynamic axis index for the K axis
8	coordSys	xdr_string	lenCoordSys	Coordinate system code

Item 1: nCells
This variable contains the number of cells in the component.

Item 2: nCellI
This variable contains the number of cells along the I (x, or radial) axis.

Item 3: nCellJ
This variable contains the number of cells along the J (y, or azimuthal) axis.

Item 4: nCellK
This variable contains the number of cells along the K (axial) axis.

Item 5: dynAxI
This variable contains the index of the relevant dynamic axis structure for the I axis. Input 0 for NONE, 1 for the first structure, etc.

Item 6: dynAxJ
This variable contains the index of the relevant dynamic axis structure for the J axis. Input 0 for NONE, 1 for the first structure, etc.

Item 7: dynAxK
This variable contains the index of the relevant dynamic axis structure for the K axis. Input 0 for NONE, 1 for the first structure, etc.

Item 8: CoordSys
This variable contains the code for coordinate system. Valid codes are
CART3D Cartesian (x,y,z)
CYL3D Cylindrical (r,θ,z)

4.2.3.3.2. 3D Parameter Arrays

Item No.	Name	Type	Length	Description
1	fI	xdr_double	nCellI +1	Coordinates of the cell faces on I
2	fJ	xdr_double	nCellJ +1 or nCellJ	Coordinates of the cell faces on J
3	fK	xdr_double	nCellK+1	Coordinates of the cell faces on K
4	grav	xdr_double	nCellK +1	Gravity cosines for cell faces on K

Item 1: fI
This array contains the coordinates of each face along the I (radial, or x) axis.

Item 2: fJ
This array contains the coordinates of each face along the J (azimuthal, or y) axis.

Item 3: fK
This array contains the coordinates of each face along the K (axial) axis.

Item 4: grav
This array contains the cosine of gravity vector for each face along the K axis.

4.2.4. Junction Sub-Block

Item No.	Name	Type	Length	Description
1	junId	xdr_long	1	Identifier for the junction
2	jCellI	xdr_long	1	I-axis-junction coordinate
3	jCellJ	xdr_long	1	J-axis-junction coordinate
4	jCellK	xdr_long	1	K-axis-junction coordinate
5	jFace	xdr_bytes	1	Face code for junction

Item 1: junID
This variable contains the TRAC numerical identifier for this junction.

Items 2-4: jCellI, jCellJ, jCellK
These variables contain the cell coordinates where the junction occurs. One- and two-dimensional variables should set the unused indices to 0.

Item 5: jFace
This variable contains a code for which face the junction attaches to. The codes are

I = Increasing I axis (downstream or increasing cell numbers).
i = Decreasing I axis (upstream or decreasing cell numbers).
J = Increasing J axis.
j = Decreasing J axis.

K = Increasing K axis.
k = Decreasing K axis.
C = Attaches at center of cell (no face).

Note: Only the first junction may attach to the "upstream" face on a 1D component; all others attach at the downstream face. (The one exception is Plenum components, where the faces can be divided into either group). Face noding always proceeds away from primary leg source and goes from the primary leg down the side legs, even if the Tee legs are nominal sources.

4.2.5. Leg Sub-Block

Item No.	Name	Type	Length	Description
1	sCell	xdr_long	1	First cell that is part of the leg
2	eCell	xdr_long	1	Last cell that is part of the leg
3	jCell	xdr_long	1	Cell to which sCell attaches

Item 1: sCell
This variable contains the starting or first cell that is part of the leg.

Item 2: eCell
This variable contains the ending or last cell that is part of the leg.

Item 3: jCell
This variable states where to connect the leg on the main tube.

Note: All 2D legs break at j = value (i.e., z = const.), and all 3D legs break at k = value (i.e., z = const.)

4.2.6. Auxiliary Component Structure Block

Auxiliary component structures allow customizing of information within a generic type. Because components generally are classed by their primary dimension, adding special information, such as signal variable type, can be a problem. The solution is to use the auxiliary component structure. This is a self-typing (not self-describing) structure that can be bypassed if the reader has not been programmed to understand the component type.

4.2.6.1. Generic Start of an Auxiliary Structure Block

Item No.	Name	Type	Length	Description
1	AuxStrT	xdr_string	lenAuxStrT	Repeat of Aux structure name
2	AuxStrRev	xdr_long	1	Revision number of Aux structure
3	AuxStrLen	xdr_long	1	Remaining length of Aux structure

4.2.6.2. PlenAux Structure Block

Currently, the only defined auxiliary structure is the "PlenAux" structure, which contains the computational length of each junction leg. (There is no cell that corresponds.)

Item No.	Name	Type	Length	Description
1	auxLab	xdr_string	lenAuxLab	Repeat of Aux structure name
2	plAxRev	xdr_long	1	Revision number of PlenAux structure
3	plAxLen	xdr_long	1	Remaining length of PlenAux structure
4	plJnDx	xdr_float	nJun	Junction length

4.2.7. Variable Definition Block

Item No.	Name	Type	Length	Description
1	varName	xdr_string	lenVarName	Short name for variable
2	varLabel	xdr_string	lenVarLabel	Descriptive label for variable
3	uType	xdr_string	8	TRAC-units-type identifier
4	uLabel	xdr_string	lenULabel	Local units label (e.g., “m/s”)
5	dimPosAt	xdr_string	lenDimPosAt	Variable dimension / position attribute
6	freqAt	xdr_string	lenFreqAt	Variable frequency attribute
7	cMapAt	xdr_string	lenCMapAt	Variable color map attribute
8	vectAt	xdr_string	lenVectAt	Variable vector attribute
9	spOptAt	xdr_string	lenSpOptAt	Variable special options attribute
10	vectName	xdr_string	lenVectName	Name of the vector association
11	vTmpl	xdr_long	1	Index of the corresponding template
12	vLength	xdr_long	1	Numerical length of the variable

Item 1: varName
Short name of variable for index and quick reference (e.g., rhol).

Item 2: varLabel
Longer descriptive name/label for this variable.

Item 3: uType
Unit type identifier from TRAC (e.g., luden).

Item 4: uLabel
Label for use in graphs and printouts (e.g., kg/m^3). This is particularly suited to smaller data-parsing scripts where the TRAC units types are not built in.

Items 5-9: Variable Attributes
The variable attributes specify the nature of the variable. All necessary variable features are covered, from dimension to color mapping to vector properties. Each attribute and its associated codes are supplied below. Each attribute is contained in its own character array that is space delineated if appropriate.

Item 10: vectorName
If a vector attribute is supplied for a vector association, this field contains the name of the vector association. If the vector attribute is NA, then this string is present in the file but may be NULL.

Item 11: vTmpl
This variable contains the index of the template that corresponds to this variable and should be used for the GUI display.

Item 12: vLength
This variable contains the number of elements output per edit for this variable. It is accurate for all variables except dynamically dimensioned variables of type "DD" (as of TRAC-M 3.450, there are none).

Attribute	Value	Description
DimPos	This gives both the dimension and cell position of the variable.	
	0D	Scalar value (not an array)
	1dCc	1D (linear) array with values at cell centers
	1dFa	1D (linear) array with values at cell faces
	2dCc	2D array [indexed as (i,j)]; values at cell centers
	2dFaI	2D array [indexed as (i,j)]; values at cell faces along I axis
	2dFaJ	2D array [indexed as (i,j)]; values at cell faces along J axis
	3dCc	3D array [indexed as (i,j,k)]; values at cell centers
	3dFaI	3D array [indexed as (i,j,k)]; values at cell faces along I axis
	3dFaJ	3D array [indexed as (i,j,k)]; values at cell faces along J axis
	3dFaK	3D array [indexed as (i,j,k)]; values at cell faces along K axis

Attribute	Value	Description
Frequency	This provides the frequency of output in the graphics file. Later, options for reduced resolution may be specified (e.g., every other edit, every third edit, etc.).	
	TI	Time-independent value (output on first edit only).
	TD*	Time-dependent value (output on every edit).
Color Mapping	This specifies the color set to use for the visualization.	
	WC*	Use water colors (blues).
	HC	Use hot colors (reds).
Vector	These codes are used to specify intrinsic and artificial vectors, matrices and tensors. These codes have an impact on the actual length output for the variable.	
	NA	There is no vector association or definition.
	M<n>	A n x n matrix. At each location specified, n**2 values are output.
	MN <m> <n>	A m x n matrix. At each location specified, m*n values are output.
	V2	A 2D vector. At each location specified, two values are output for i and j axes. Name of vector is the same as the variable name.
	V3	A 3D vector. At each location specified, three values are output for i, j, and k axes. Name of vector is the same as the variable name.
	VI <name>*	i component of vector valued function (i,j,k must be sequential). This name appears in the vectors submenu of XTV. The rank of the vector is equal to the rank of the component. (Thus, 2D components will not have VK defined.) The vector exists at as many cells as are defined by the length attribute of the component.
	VJ	j component of vector valued function (i,j,k must be sequential).
	VK	k component of vector valued function (i,j,k must be sequential).P
	IT <name>*	i row of tensor function (i,j,k must be sequential). This name appears in the vectors submenu of XTV. The rank of the tensor is given by the component type. Thus, a scalar tensor of a 3D component has three values.
	JT	j row of tensor function (i,j,k must be sequential).
	KT	k row of tensor function (i,j,k must be sequential).
Special Options	These codes are special purpose options (see the individual explanations).	
	ID	(inset display) Display as an inset value (as wall temperatures are now).
	UV	(unlisted value) Do not place on variable selection list (typically a variable used for dimensioning dynamic arrays).

5.0. REFERENCES

1. B. T. Adams, J. F. Dearing, P. T. Giguere, R. C. Johns, S. J. Jolly-Woodruff, J. W. Spore, R. G. Steinke, J. Mahaffy, and C. Murray, "TRAC-M/Fortran 90 (Version 3.0) Programmer's Manual," Los Alamos National Laboratory report LA-UR-00-803 (February 2000).

2. R. G. Steinke, V. Martinez, N. M. Schnurr, J. W. Spore, and J. V. Valdez, "TRAC-M/Fortran 90 (Version 3.0) User's Manual," Los Alamos National Laboratory report LA-UR-00-834 (February 2000).